Performance Analysis of Production POP Runs on the Cray XT3

A Case Study

James B. White III (Trey)
trey@ornl.gov

Acknowledgement

Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

Motivation

- High-resolution POP runs dominate two INCITE projects
  - Will continue to be important
- Performance tuning of actual production run on actual production hardware
- Acceptance test for new supercomputers
- Regression test for upgrades

Test case

- Snapshot of production ocean spin-up run ongoing by Mat Maltrud of LANL
  - Input and restart files from a representative job
- 0.1° global resolution
  - 3600 x 2400 x 42 grid
  - Tripole grid, unlike available benchmarks
- 1 simulated day
- Representative I/O
  - 38 GB of input and restart files
  - 43 GB of history and checkpoint

Process

- Create a new test case in its own run directory
- Build and run
- Compare to previous runs
- Commit to a Subversion repository
  - Large input files backed up to HPSS, not in repository
  - Large output files not kept
- Decide what to do in next test

Batch script

Calculate size as twice the number of cores - *only works for even core counts*

Set stripe widths for output directories (and thus files)*

Add core count to input namelist

Small pages

```
#!/bin/ksh -l
#PBS -A STF006
#PBS -q debug
#PBS -N pop
#PBS -l walltime=0:30:00
#PBS -j oe

export CORE_PATH=$PBS_O_WORKDIR
cd $PBS_O_WORKDIR
(( NCPU=2 * $PBS_NNODES ))
mkdir movie
lfs setstripe movie 0 -1 -1
mkdir restart
lfs setstripe restart 0 -1 -1
mkdir tavg
lfs setstripe restart 0 -1 -1

if [[ ! -x pop.$NCPU ]]
then
   /bin/ls pop.$NCPU
   exit $?
fi

sed "s/XXXX/$NCPU/" pop_in.sed > pop_in

yod -small_pages -sz $NCPU pop.$NCPU
```

* Steve Gottleib noticed that the last “lfs” command should set “tavg”, not “restart”. All performance results presented here include this typo in the script, resulting in a striping of just four for the “tavg” output.

Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

First test

- 360 cores (fits in “debug” partition)
- Compute time of 843 seconds
- Total time of 1990 seconds
- 58% of runtime doing init and I/O

POP I/O

- Production jobs currently use one I/O process
- POP allows multiple I/O processes
  - Read and write contiguous elements of global 3D data structure to a single file
  - Aggregated horizontal slabs
  - Parallel across vertical levels
  - 42 vertical levels, up to 42 I/O processes
- Not often used
  - Some file systems don't support it

Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

I/O tests

- 2 I/O processes
  - Similar runtime
  - Identical standard output
- 42 I/O processes
  - I/O and init down from 1147 seconds to 238 seconds
  - Identical standard output
  - But large binary output files have slightly different sizes!
- **To do:** debug POP parallel output

Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

Minor bug fixed

- Junk in standard output

```
(ttd_mod:ttd_init) Reading TTD surface regions from
../in/grid/ttd_8patches.r8^@^@^Y^@^@^@P]6/^@^@^@^@350^A^@^@^@^@^@^@^
B^@^@^@^@^@^@^@320^@331)^@^@^@^@360377330)^@^@^@^@P357212.^@^@^@^@20
0212330)^@^@^@^@360377330)^@^@^@^@^P210277377^@^@^@^@^Q322f^@^@^@^@^
@200332252)^@^@^@^@p^DS)^A^@^@^@320^@331)^@^@^@^@360377330)^@^@^@^@P
357212.^@^@^@^@240213277377^@^@^@^@200210277377^@^@^@^@8326f^@^@^@^@
^@^@^@^@^@^@^@^@350^A^@^@^@^@^@^@H277247^]^@^@^@^@?^@^@^@^@^@^@^@?
^@^@^@^A^@^@^@(277247^]^@^@^@
```

- The culprit: initialization of Fortran character string “`region_filename`”

```
cindx2 = len_trim(ttd_region_file)
region_filename(1:cindx2) = trim(ttd_region_file)
```

What’s the bug?

Minor bug fixed

```
cindx2 = len_trim(ttd_region_file)
region_filename(1:cindx2) = trim(ttd_region_file)
```

- First assignment to “`region_filename`”
- Rest of variable is undefined
- Easy fix: direct assignment (no substring)
- Fortran does what you’d want
  - “`region_filename`” shorter? Truncate!
  - “`region_filename`” longer? Fill with spaces!
- Fix:

```
region_filename = trim(ttd_region_file)
```

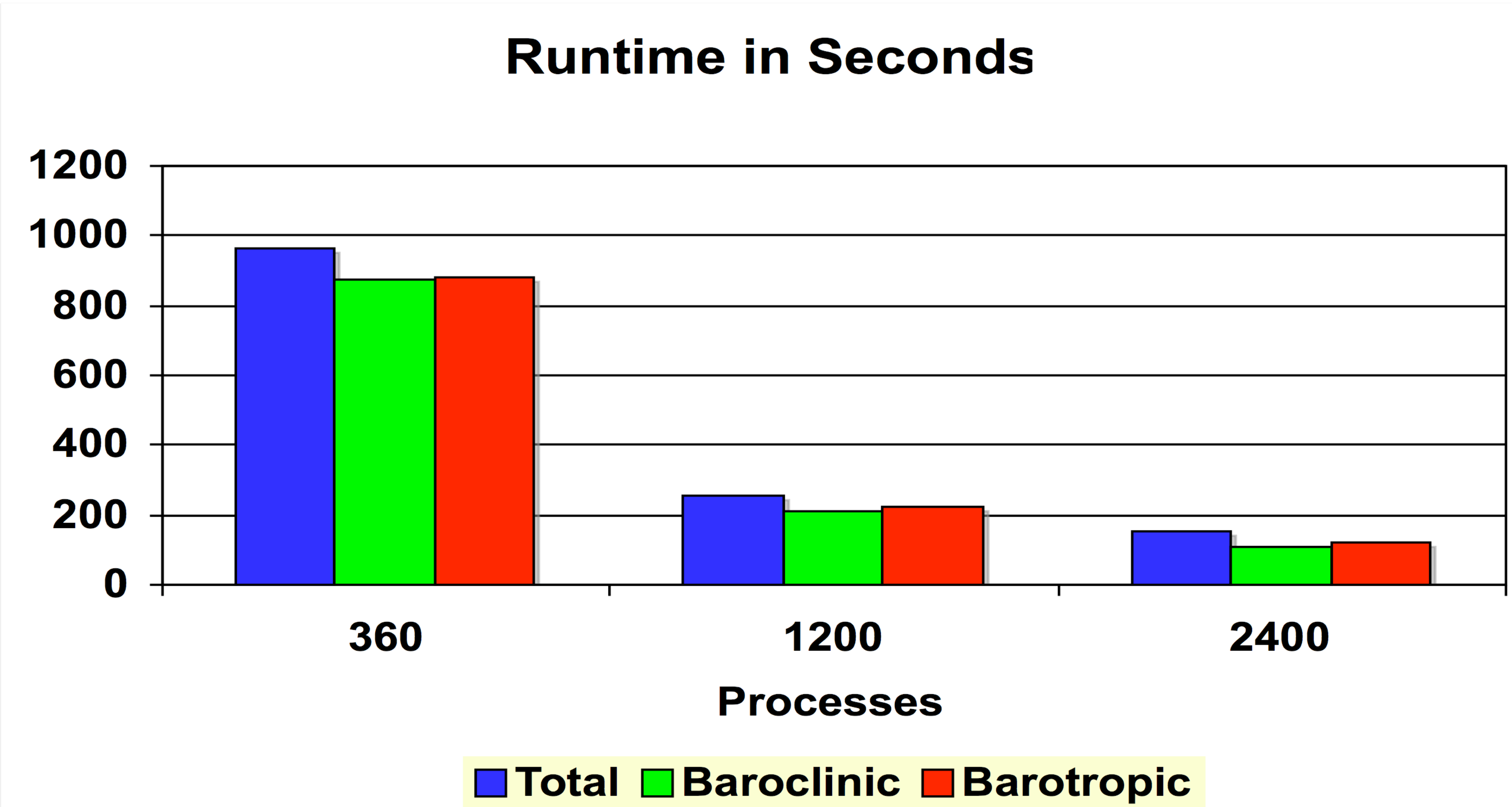
Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

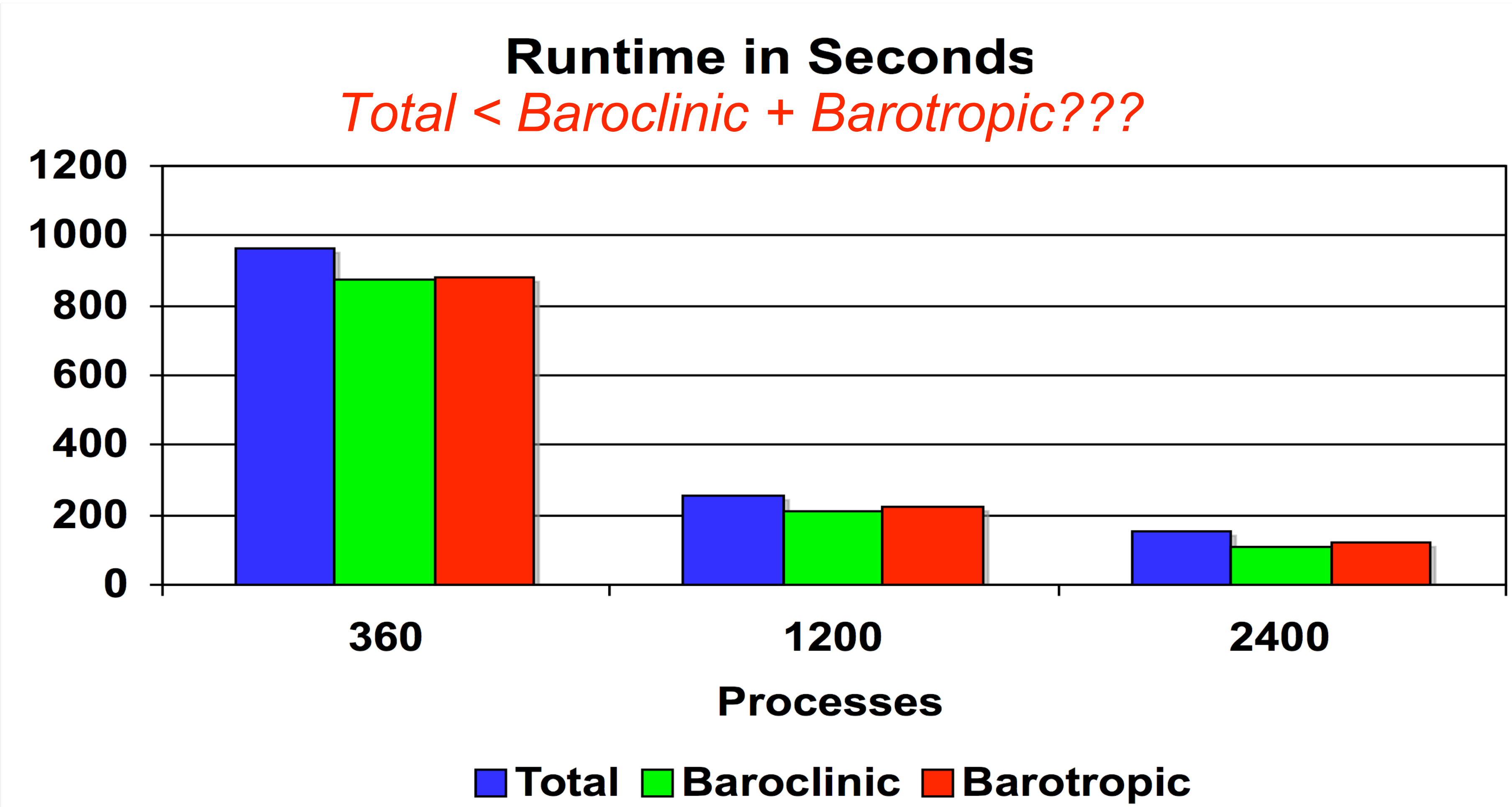
Scaling runs

- Scaling of computation time
  - 360, 1200, 2400
- Computation split into two phases
- Baroclinic
  - 3D, explicit, nearest neighbor
  - Dominates at small process counts
- Barotropic
  - 2D, implicit, conjugate gradient
  - Latency bound, limits scaling
- Built-in timers

POP performance (according to built-in POP timers)



POP performance??? (according to built-in POP timers)



Timer output

```
Timer number 11 Time =      155.23 seconds   STEP
  Timer stats (node): min =       155.21 seconds
                      max =       155.23 seconds
                      mean=       155.23 seconds
…
Timer number 12 Time =      105.91 seconds   BAROCLINIC
  Timer stats (node): min =         1.91 seconds
                      max =       105.91 seconds
                      mean=        76.92 seconds
…
Timer number 13 Time =      123.60 seconds   BAROTROPIC
  Timer stats (node): min =        23.90 seconds
                      max =       123.60 seconds
                      mean=        50.33 seconds
```

*Each timer reports **maximum** among processes*

And there's some major load imbalance

Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

Try CrayPAT

```
pat_build -u -g
stdio,io,math,mpi,system

pat_report -O
ca+src,heap,load_balance,mpi,program_ti
me,read_stats,write_stats
```

- "`.xf`" files get very large, increasing with process count
- "`pat_report`" failed with report from 4800 processes

Dominant cost on 2400 processes

```
 Time % |   Cum.  |        Time |       Calls |Experiment=1
        | Time %  |             |             |Group
        |         |             |             | Function
        |         |             |             |  Caller
        |         |             |             |   PE='HIDE'

 100.0% | 100.0% | 268.546663 | 4540225188 |Total
|-------------------------------------------------------
|  76.3% |  76.3% | 204.813429 |  233759543 |MPI
||------------------------------------------------------
||  49.8% |  49.8% | 102.050754 |    7841974 |mpi_allreduce_
|||-----------------------------------------------------
|||  57.8% |  57.8% |  59.034285 |    6095952 |global_reductions_global_sum_nfie
lds_dbl_:global_reductions.f90:line.287
||||----------------------------------------------------
||||  72.4% |  72.4% |  42.733456 |      39552 |solvers_pcg_chrongear_:solvers.f
90:line.734
```

First reduction in barotropic conjugate-gradient solver

Load imbalance on 2400 processes

```
 Time %  |  Cum.  |         Time |       Calls |Experiment=1
         | Time % |              |             |Group
         |        |              |             | Function
         |        |              |             |  PE[mmm]

 100.0%  | 100.0% | 268.546663 | 4540225188 |Total
|----------------------------------------------------------------
|  76.3% |  76.3% | 204.813429 |  233759543 |MPI
||---------------------------------------------------------------
||  49.8% |  49.8% | 102.050754 |    7841974 |mpi_allreduce_
|||--------------------------------------------------------------
|||    0.7% |    0.7% | 232.130222 |      25371 |pe.214
|||    0.1% |   86.7% |  28.416951 |      25371 |pe.271
|||    0.0% |  100.0% |   3.394145 |       2335 |pe.309
|||==============================================================
```

Sure enough, there's a load imbalance

Now what?

- Load imbalance in first reduction of barotropic solver
- What's the distribution among processes?
- And what's the distribution in time?
  - Are all the calls to the solver imbalanced?
  - Just the first one, or just a few?
- A trace of the calls would tell
  - If you could store it all
  - And if you could wade through it for the useful info
- But even a profile is too big at high process counts

Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

OK, they're just timers

- Instrumented solver and nearby calls with "special" timers
- Timers accumulate multi-resolution histograms of the measurements
- Output for each process
- Planned automatic reduction of per-process output, to be reported at CUG 2007
- More fun than reading CrayPAT docs and figuring out how CrayPAT can probably solve the same problem

Timing results with 2400 processes

- Added a barrier before first reduction (per iteration)
  - Distinguish between active “reducing” and mere “waiting”
- Barrier dominated, with bimodal timing among processes
  - 1797 processes spent less than 3 seconds
  - 603 processes spent more than 75 seconds
- But when did this time accumulate, at one long iteration or lots of average-sized iterations?

Timing results with 2400 processes

- Short example (one of 1797)

```
Timer #014, pcg_chrongear mpi_barrier
       20s of ticks:          128
```

- Long example (one of 603)

```
Timer #014, pcg_chrongear mpi_barrier
       30s of ticks:            1
      500s of ticks:           85
      600s of ticks:           42
```

- Some processes consistently wait a long time, while most wait little

Culprit: Cartesian distribution

- Each process gets one block
  - Some blocks are all land, no work
  - Collectives are over all blocks
  - Land-locked blocks are always waiting
- POP also has “balanced” distribution
  - Requires more blocks than processes
  - Blocks are load balanced across processes
  - More communication cost from increased surface to volume
  - Needs debugging on XT
- New distributions are easy to create
  - Each process keeps all distribution info (not strictly scalable)

Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

It's just "pack"

- Like Fortran "`pack`"
- Pack non-empty blocks on processes
  - One per process
- Use fewer processes than total blocks
- ***The Price is Right***, in reverse
  - As few processes as possible, without going under
- Upper bound on performance improvement ~30%
  - Related to fraction of Earth's surface that is land
- Should help timers make sense

```
function create_distrb_pack(nprocs, work_per_block) result(dist)
  ! Pack blocks with nonzero work onto processors in block-number order.
  implicit none
  type(distrb) :: dist
  integer(int_kind) :: nprocs
  integer(int_kind) :: work_per_block(:)
  integer :: b, bpp, i, n, nblocks, p

  nblocks = count(work_per_block(:) /= 0)
  bpp = ((nblocks-1)/nprocs)+1
  n = size(work_per_block,1)
  call create_communicator(dist%communicator, nprocs)
  dist%nprocs = nprocs
  allocate(dist%proc(n))
  allocate(dist%local_block(n))
  p = 1
  b = 1
  do i = 1, n
     if (work_per_block(i) /= 0) then
        dist%proc(i) = p
        dist%local_block(i) = b
        b = b+1
        if (b > bpp) then
           p = p+1
           b = 1
        end if
     else
        dist%proc(i) = 0
        dist%local_block(i) = 0
     end if
  end do
end function
```

Cartesian verus packed

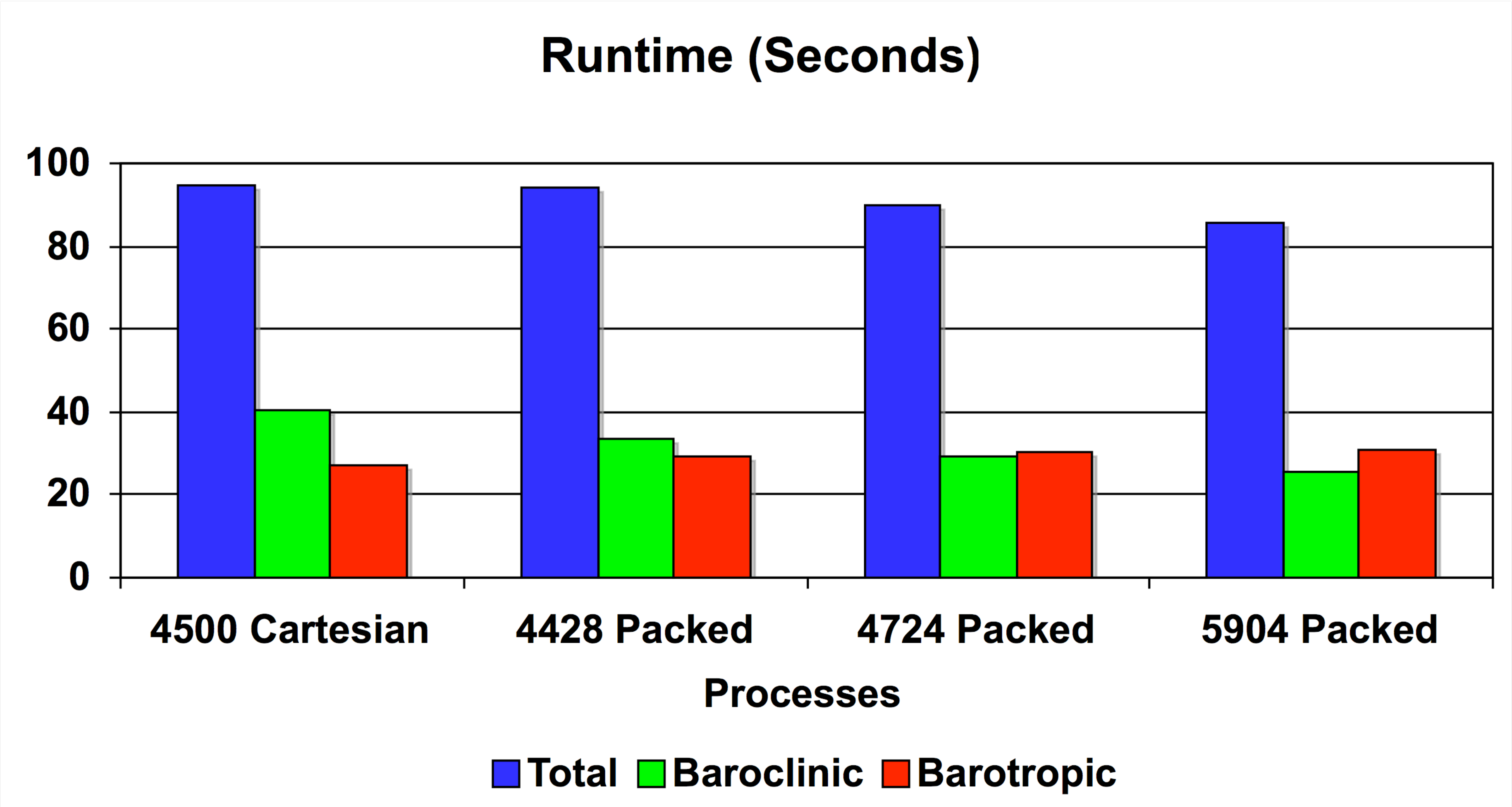
- 360 Cartesian processes
- 853 seconds total
- Baroclinic
  - 755 seconds max
  - 607 seconds mean
  - 7 seconds min
- Barotropic
  - 764 seconds max
  - 182 seconds mean
  - 70 seconds min

- 303 packed processes
- 844 seconds total
- Baroclinic
  - 754 seconds max
  - 736 seconds mean
  - 642 seconds min
- Barotropic
  - 78 seconds max
  - 71 seconds mean
  - 69 seconds min

Large runs

Baroclinic still shrinking

Barotropic growing!



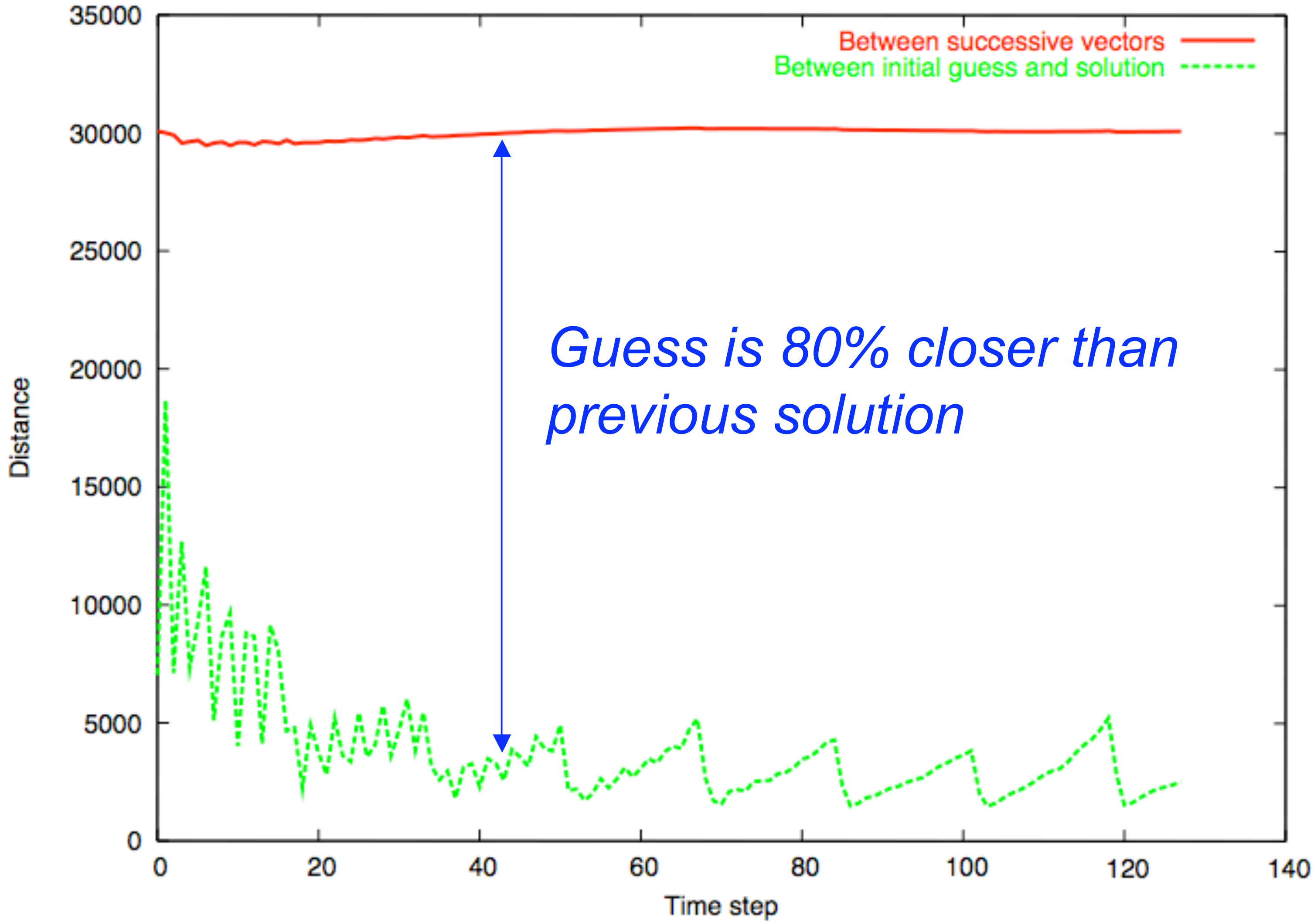
Barotropic growing

- Conjugate-gradient solver
- Dominated by `MPI_Allreduce`
  - Like POP benchmarks
- Reduce cost be reducing iterations
- Better initial guess?
- Better preconditioning?

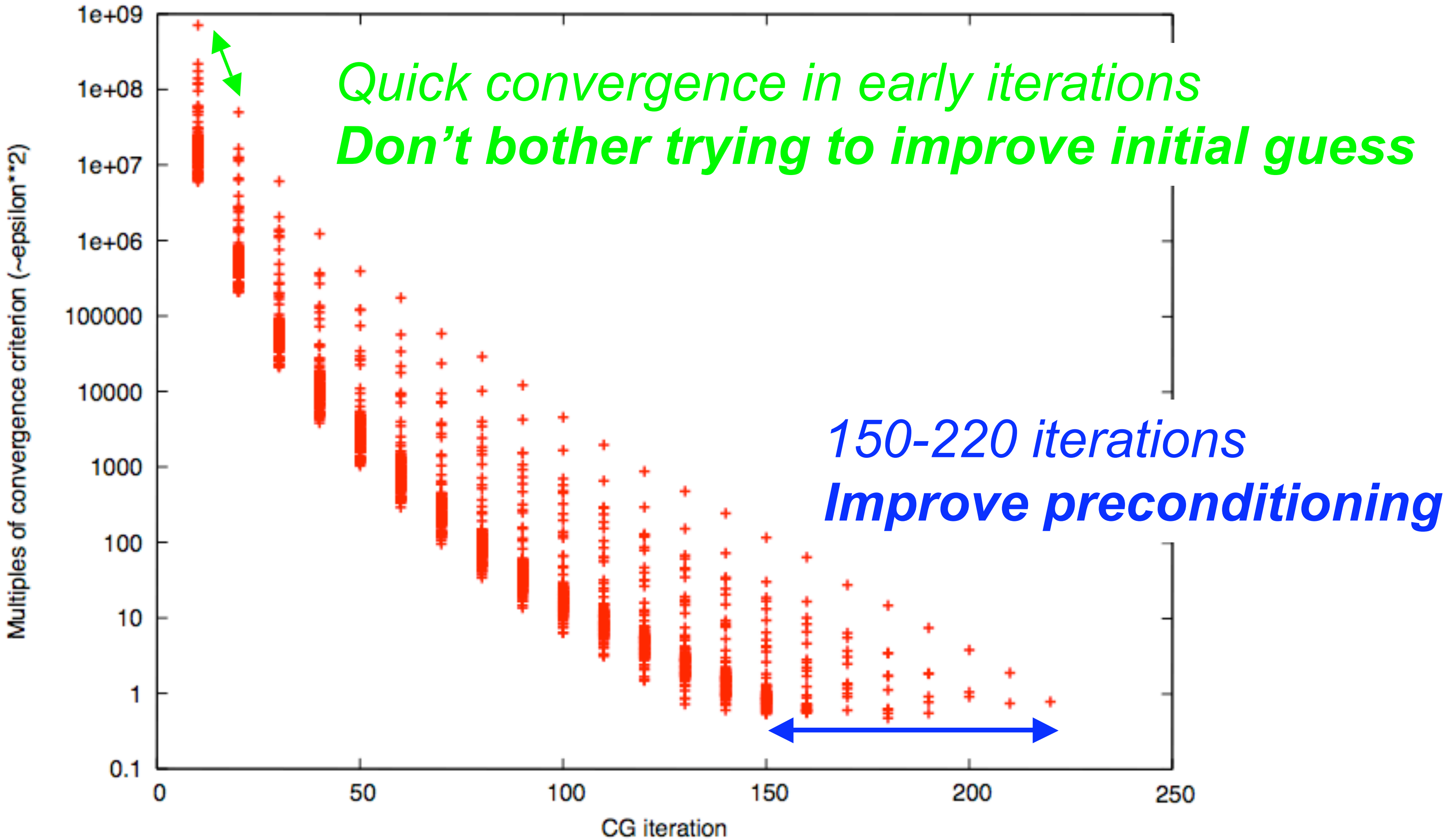
Timeline

- First test
- I/O tests
- Minor string bug
- Scaling runs
- CrayPAT analysis
- TreyPAT analysis
- New decomposition
- Solver convergence

Quality of initial guesses



Convergence rates



Summary of production-POP performance analysis

- Barotropic CG solve limits scaling (as in benchmarks)
  - Good early convergence, don't bother with improving initial guesses
  - Slow late convergence, try to improve preconditioning
  - **New preconditioner results by next CCSM Workshop (June)**
- Trouble with timers
  - Load imbalance obfuscates min/max timers
  - CrayPAT output grows with process count
  - **CUG 2007 talk on multi-resolution timing with parallel reduction**
- I/O
  - Striping essential
  - **Need to debug multiple writers**